POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Master Thesis

Pin Control Defense

Protecting Linux Kernel Pin Control Subsystem from Pin Control Attack

Supervisor
prof. Antonio Lioy

Candidate
Andrea Genuise

Academic Year 2016-2017

† A nonno Carmelo

Summary

Recently embedded systems have become more and more integrated with all aspects of our lives, and their security concerns have risen as well. They spread in various fields such as automotive, electronic devices, home automation, manufacturing and mission critical applications. These systems, in particular PLCs deployed within the context of an Industrial Control System, use Input/Output interfaces to interact with the physical world by means of sensors and actuators. As demonstrated by a novel kind of attack called Pin Control Attack, one can tamper with the integrity or the availability of legitimate I/O operations, factually changing how a PLC interacts with the outside world and possibly causing physical damage to people and environment. In this thesis we design a possible countermeasure to the attack and implement it for Linux Kernel on an ARM-based Programmable Logic Controller, showing its effectiveness and impact on PLCs which usually have very limited resources and strict timing requirements.

Acknowledgements

TODO Acknowledgements.

Contents

Chapter 1

Introduction

Embedded systems are widely used today in various applications, from cars, cell phones, home automation, to critical infrastructures like power plants and power grids, water, gas or electricity distribution systems and production systems for food and other products.

Since they were almost isolated from the network, embedded systems have not been designed and built with security concepts in mind. However, many recent cyber-attacks demonstrated that such an assumption is no more valid, and the security of embedded systems became an open question to deal with.

Unfortunately, this could be a more challenging problem with respect to security for desktop and enterprise computing, for the following reasons:

- the limited capabilities and the strict timing requirements these systems have;
- the physical side effects a security breach may lead to, e.g. property damage, personal injury, death and even environmental or nuclear disaster.

In the next sections we first describe a particular type of embedded systems, then present the problem on which the rest of the paper is focused and clarify the goal of this thesis.

1.1 Programmable Logic Controllers

Within the context of an Industrial Control System (ICS), embedded systems are better known as Programmable Logic Controllers (PLCs). PLCs are special-purpose embedded systems, usually deployed into harsh environments to control critical processes, e.g. automotive systems, assembly lines, robotic devices or any industrial machine that requires high control precision and reliability.

A simplified version of the environment in which a PLC may operate is outlined in Fig. 1.1.

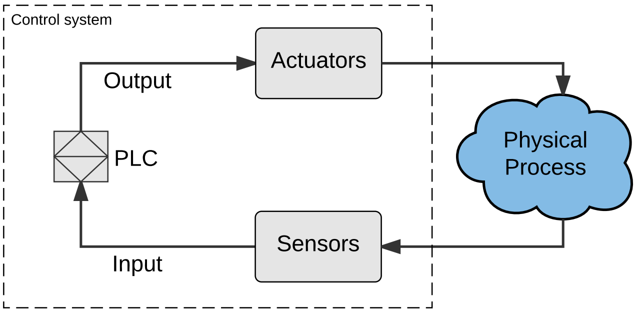


Figure 1.1: PLC environment

The PLC interacts with the physical world through external components called *sensors* and *actuators*. The sensors are responsible for reporting measurements about a set of properties of the physical process, while the actuators, as the name suggests, acts on the process modifying its current state.

The main task of the PLC is contained into the control program (the *logic*) which is programmed and loaded by the industrial operator or supervisor. The logic is executed repeatedly as long as the controlled system is running. Each single execution is known as *scan cycle*. For each scan cycle, the PLC reads the current state of the physical process measured by sensors. Internally, the logic takes these values as input and performs some calculations based on the loaded control algorithm. The output of the logic is then sent to the actuators, thus modifying the state of the controlled process.

The PLC main scan cycle is shown in Fig. 1.2.

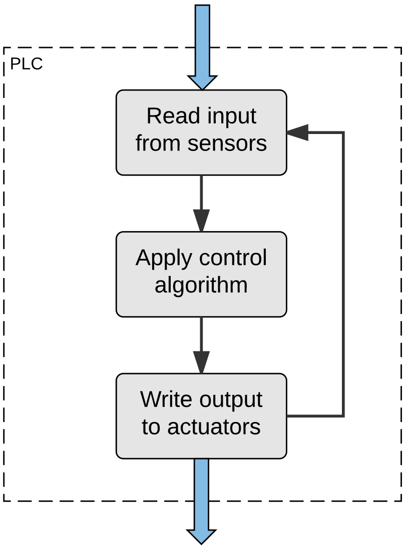


Figure 1.2: PLC logic main scan cycle

Based on the nature of the controlled process, the logic must satisfy different safety and timing properties. Furthermore, the majority of the PLCs are provided with a real-time operating system, in order to guarantee that the strict timing requirements are always respected. A violation of these requirements may lead to an unsafe behaviour of the physical process, and the consequences are heavily related to the nature of the process under control.

1.2 Problem statement

From a computer engineering perspective, PLCs control the outside world via their Input/Output (I/O) interfaces, also known as *pins*. Therefore, it is crucial that they are both reliable and secure. I/O pins are accessible through a specific memory region, called I/O memory. The I/O memory includes a set of registers through which the I/O can be configured.

The problem relies on the fact that there is neither any memory protection mechanism nor any hardware interrupt designed for restricting the access to these memory addresses. They can be easily written by a malicious application even while the system is running, actually changing the behaviour of the I/O in a dramatic way. In [2], Abbasi et al. showed how this feature is exploitable by attackers, who can tamper with the integrity or the availability of legitimate I/O operations, factually changing how a PLC interacts with sensors and actuators. Based on these observations, they introduced a novel attack technique against PLCs, which they call Pin Control Attack (also I/O attack). The salient features of this new class of attacks are:

1. It is intrinsically stealth. The alteration of the pin configuration does not generate any interrupt, preventing the Operating System (OS) to react to it.
2. It is entirely different in execution from traditional techniques (e.g. manipulation of kernel structures or system call hooking) which are typically monitored by off-the-shelf protection systems.
3. It is viable. It is possible to build concrete attack using it.

To better understand a possible attack scenario, Fig. 1.3 shows a simplified control system.

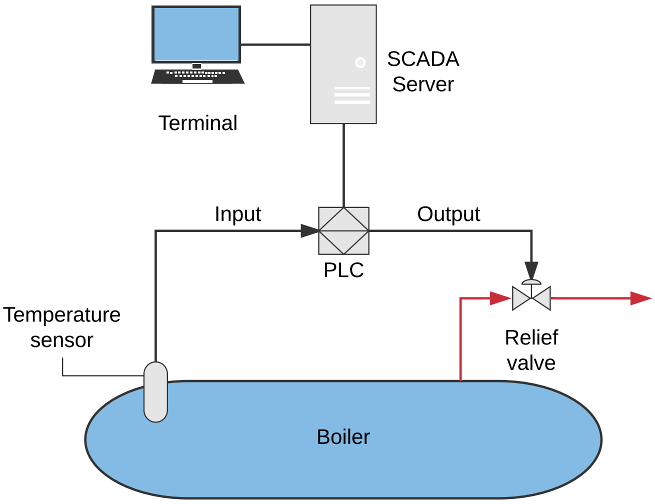


Figure 1.3: Simplified control system: a possible target of Pin Control Attack.

The system consists of a boiler equipped with a relief valve driven by the PLC according to the value provided by a temperature sensor. The PLC is connected to a SCADA server (Supervisory Control And Data Acquisition) to keep trace of its operation. The server is accessible from a terminal, through which an operator can load the logic into the PLC and monitor its current state.

Suppose that the PLC logic is programmed to read the values from the temperature sensor, and to open the relief valve if the temperature goes over $80°C$. An attacker could tamper the temperature value read by the PLC, reconfiguring the pin related to the sensor as output and writing its own desired value (e.g. always $50°C$) regardless of the actual temperature. Even simpler, one could reconfigure the pin connected to the valve (e.g. setting it as input) factually disabling any eventual command to the actuator.

Actually there is no detection mechanism able to react to these configuration changes, neither in the PLC firmware nor in the control system. Moreover, the operator of the control system will not be able to see the real temperature nor the actual valve state from the terminal, so it may likely not notice the intrusion at all. Such a condition may lead to an uncontrolled overheating, and if it is not detected in time it could even make the boiler to explode.

1.3 Aim of the thesis

The goal of this thesis is to design and develop a defensive mechanism against Pin Control Attack, extending the pin controller in order to be able to detect it. It is a challenging task for two reasons:

1. The pin control mechanism lacks hardware interrupts, so it is not possible to directly react to any configuration change. More complex detection mechanisms are needed in order to achieve the highest possible detection rate.

2. The resources available within an embedded system like a PLC are extremely limited. Therefore, our solution must be extremely agile and light since the smallest delay in the PLC I/O operation may have unintended consequences for the controlled process.

The work has been divided into three main steps. First, we discuss the idea behind Pin Control Attack, analyse its threat model and different implementations on an Embedded Linux ARM-based architecture. Second, based on the assumptions contained in the threat model, we describe and evaluate the design and the implementation of our defense. Finally, we validate the proposed architecture on the target system.

The target of our implementation is a real-time version of the Linux kernel running on an ARM-based SoC, as this is one of the solutions actually used for mid-range embedded systems, in particular PLCs. In the context of the Linux kernel, the pin controller is known as Pin Control Subsystem [3]. The implementation will be validated with respect to the following parameters: detection rate and performance overhead.

A brief organisation of the rest of the thesis follows in the next section.

1.4 Organisation

In Chapter 2 we list the known attacks to embedded systems existing in the literature, and then discuss the off-the-shelf protection mechanisms. To the best of our knowledge at the time of writing, as Pin Control Attack is a new kind of attack, none of the existing protections is able to prevent nor detect it. Chapter 3 contains, after a brief description of the embedded systems architecture, a detailed analysis of Pin Control Attack design and implementation. In Chapter 4 we present the architecture of our proposed Pin Control Defense, then we describe the implementation and provide developer and user manual for it. The Chapter 5 provides the results obtained during the experiments, showing the detection rate and the performance overhead on a PLC environment. Finally, in Chapter 6 we analyse the shortcomings of our defense and the possible future works and improvements.

Chapter 2

Related work

In this chapter we summarise the state of the art about security of embedded systems at the time of writing. First, we discuss about the attacks of the recent years, showing how the embedded systems security concerns are rising. Next, we analyse the defense mechanisms currently available in the literature, realising that they are still in a very early stage of their life.

2.1 Attacks

In the past few years we have seen several attacks targeting embedded systems: most notably the infamous Stuxnet [4] targeting an Iranian nuclear facility in 2010. More recently Grandgenett et al. [5] analysed the authentication protocol between the RSLogix 5000 software and the PLC, based on a simple challenge-response mechanism. Since the protocol lacks freshness in its messages, is vulnerable to replay attacks, through which an attacker could repeat privileged commands to the PLC. Furthermore, they found that both the decoding of the challenge and the encoding of the response use an RSA-2048 key which is hard-coded in the RSLogix software, and it is actually valid for any Rockwell/Allen Bradley PLC. This indicates how the security mechanisms of these systems often have a really poor design, if any.

Papp et al. [6] analysed the existing attacks on embedded systems, relying on the proceedings of security conferences, with a focus on computer hacking, and on the Internet for media reports, blogs and mailing list. They built a taxonomy based on five dimensions: precondition, vulnerability, target, attack method and effect of the attack, showing that the threats to embedded systems are similar to the ones that affect traditional IT systems. However, embedded systems still lack solutions and tools to address these issues, and many ongoing research efforts are trying to deploy and adapt them to the needs of this field.

For our purpose, we may classify the attacks found in literature using a simpler criterion based on the attack method. We may distinguish three main categories:

1. **Firmware modification**: all the attacks aiming to upload a malicious firmware version (or part of it) in the device belong to this category.

2. **Logic modification**: this category consists of the attacks that modify the PLC logic in some way. In this case the integrity of the firmware is not violated, but a malicious program, or logic, is inserted into the PLC to make it misbehave during the control process.

3. **Control flow modification**: it includes the attacks that alter the normal control flow of a process by leveraging classic programming vulnerabilities such as buffer overflow or expired pointer dereference.

We briefly report about these different kind of attacks in the following sections.

2.1.1 Firmware modification attacks

Almost all modern embedded systems provide a way to update the firmware, and the attackers could exploit this feature to upload its own malicious firmware. Basnight et al. [7] reverse engineered an Allen-Bradley ControlLogix L61 PLC firmware showing how to bypass the firmware update validation method and successfully upload a counterfeit firmware. Peck et al. [8] demonstrate how using commonly available software an attacker can write and load his malicious firmware into Ethernet cards of devices used in control systems, potentially infecting the entire industrial control system. Cui et al. [9] discovered a vulnerability in the HP-RFU (Remote Firmware Update) feature of LaserJet printers, that allows remote attackers to make persistent modifications to the printer's firmware by simply printing to it.

2.1.2 Logic modification attacks

Stuxnet [4] belongs to this category. Along with its several components, mainly used to replicate and control the malware, its core is essentially an infected version of a SCADA software library used to program the PLC itself. By hooking some of the library functions it is able to load infected code and data blocks into the PLC and hide itself from the operator. McLaughlin et al. [10, 11] evaluated some techniques and implemented a tool (SABOT) to infer the structure of a physical plant and craft a dynamic payload, allowing an attacker to cause an unsafe behaviour without having a deep *a priori* knowledge of the target physical process. Similar techniques might mitigate the precondition needed by an attack, making it even more viable. Beresford [12] showed how the PLCs and the protocols used for communication in control systems were built without any security in mind, and demonstrated that they are affected by many vulnerabilities which may also enable the attacker to know the current configuration and rewrite the PLC logic. More recently, Klick et al. [13] used an internet-facing PLC as a network gateway by prepending a backdoor, made of a port scanner and a SOCKS proxy, to the existing logic code of the PLC. They developed a proof-of-concept tool called *PLCInject* to demonstrate their approach and measure the effects on the network.

2.1.3 Control flow modification

Many recent advisories [14–17] from ICS-CERT (Industrial Control System Cyber Emergency Response Team) report about various programming vulnerabilities affecting both PLC firmwares and control softwares. Most of them allow remote code execution and could be exploited without requiring particularly high skills. The vulnerabilities discovered by Beresford [12] also allow the attacker to insert a payload into the logic and subvert the control flow to execute malicious code. Nevertheless, the majority of the PLCs run the applications with root privileges, so it is quite simple for an attacker to get a root shell. One of the most dangerous kind of control flow attacks consists of ROP (Return-Oriented Programming) techniques, or similar variants [18, 19] which leverage different sequence of instructions equivalent to a return instruction. Since code vulnerabilities may affect embedded systems [12, 14–17], ROP techniques are applicable as well. Furthermore, due to the limitations imposed by these systems, is even more challenging to defend against them.

2.2 Defenses

Even though many incidents in SCADA systems occurred in the past decades [20, 21], the scientific community, together with PLC producers (Siemens, Hitachi etc.) and antivirus producers (Symantec, Kaspersky etc.), started to explore the security concerns of these systems only after the discovery of Stuxnet malware in 2010.

Since the communication protocols are the most vulnerable area in embedded systems, many efforts have been directed to *network-based* defenses [22]. In 2013, Clark et al. [23] designed a defense scheme against Stuxnet in which commands from the system operator to the PLC are authenticated using a randomised set of cryptographic keys. Hadiosmanovi et al. [24] proposed a semantic-aware intrusion detection system, which is able to build a prediction model by observing the values of the process variables from the network communication, and then detect unauthorised changes with respect to the model.

In our work, however, we will focus on *host-based* defenses, in which the protection mechanism resides in the embedded system itself. Similarly to what we did for attacks, we can divide host-based defenses into the corresponding categories:

1. **Firmware integrity**: includes all the available techniques for preventing or detecting any malicious change in the PLC firmware.
2. **Intrusion detection**: host-based intrusion detection systems responsible for detecting rootkits or malicious software that could tamper with the normal process controlled by the PLC.
3. **Control flow integrity**: all the available mechanisms to defend against control flow attacks, like control flow integrity techniques or anti-ROP defenses, belong to this category.

2.2.1 Firmware integrity

In order to detect malicious modifications in the firmware of embedded systems, Wang et al. [29] proposed ConFirm, a low-cost technique, embeddable into the bootloader, based on measuring the number of low-level hardware events that occur during the execution of the firmware. To count these events, ConFirm leverages a set of special-purpose registers, the Hardware Performance Counters (HPCs), which readily exist in many embedded processors. The approach is divided into two phases: offline profiling and online checking. The offline profiling is executed before the system is deployed, and consists of registering the HPC signatures of code paths from a clean copy of the targeted firmware. The signature database is then embedded into the bootloader together with the ConFirm payload executed in the second phase. During the online checking, the same monitored paths are measured and compared to the golden references. Although this technique raises the bar for firmware modification attacks, if an attacker is able to reverse engineer and modify the bootloader, which usually has some update procedure as well, then the entire mechanism could be circumvented.

Other approaches that could defend against firmware modifications are based on Trusted Computing. While this kind of technology is commonly deployed in more capable systems, such as desktop or enterprise, the most of the embedded systems need a solution with lower resource requirements. The TrustZone Technology [25] enables trusted computing for ARM platforms by extending the hardware architecture, essentially the system bus, the processor core and the debug infrastructure, with security-aware components. Furthermore, Koeberl et al. [26] proposed a hardware-enforced security architecture named TrustLite, which is able to provide trusted computing capabilities on resource-constrained embedded devices without requiring CPU security extensions.

A trusted computing architecture can also enable attestation, through which a system, called verifier, can verify the integrity state of another system, called prover, which should provide a cryptographic proof of its integrity. Similar technologies could also be used to design a secure firmware upgrade mechanism. The PLC vendors need the possibility to provide firmware updates for their own devices, most likely when some vulnerabilities are discovered after release. Fuchs et al. [27] discussed the benefits of Trusted Computing Groups Trusted Platform Module (TPM) 2.0 as a security-anchor for embedded systems, and proposed a proof-of-concept implementation of advanced remote firmware upgrade for embedded systems relying on the unique features of TPM 2.0.

A different approach is proposed by Lee, B. & Lee, J. [28], which leverages blockchain technology to securely check firmware versions, validate the correctness of firmware, and download the latest firmware for the embedded devices. Even though their work is focused on embedded systems intensively inter-connected within an IoT environment, the increasing number of internet-facing controllers let us suppose that may be worth investigating whether this technology could be applied to PLCs as well.

2.2.2 Intrusion detection

The PLC logic is usually executed in a scan cycle, in which the PLC reads the inputs from the sensors, executes the logic and writes the outputs to the actuators. Zonouz et al. [30] devised an approach capable of detecting whether a PLC logic could violate physical plant's safety requirements. Their technique is based on logic binary code analysis and model checking, and could be integrated in the PLC runtime itself, so that the check is made every time a new logic is uploaded to the system. Garcia et al. [31] leveraged the advanced computational power of embedded hypervisors, that could be coupled with modular embedded controllers, to provide both a memory verification solution and an intrusion detection system from within the PLC itself. The embedded hypervisor provides a library directly accessible from the PLC scan cycle either synchronously or asynchronously, and the hypervisor and the PLC can communicate through shared memory. Their approach may be extended to integrate any kind of security solutions inside the PLC. Cui et al. [32] proposed a host-based defense mechanism called Symbiotic Embedded Machines (SEM), designed for injecting intrusion detection functionalities into embedded system firmware code. The injected Symbiotes are basically code structures that will co-exist with the legacy firmware, sharing computational resources with it while protecting it against code modification. The SEM injection process is randomised, so that the payload is divided into slices at random locations, called control-flow intercepts, and are activated when the firmware execution reaches these points. In this way SEM executes itself alongside the original OS while remaining stealthy.

Moreover, Trusted Computing could also be used to enable malicious code detection capabilities at runtime. The main problem with this technology is to deploy it into embedded systems without impacting its limited resources and real-time constraints. Seshadri et al. [33] proposed a software-based attestation technique, named SWATT, to verify the memory contents of embedded devices and establish the absence of malicious code through a challenge-response protocol. SWATT is designed in a way that the minimum change to the code will result in a detectable delay. However, further research [34] has shown that this time-based approach is hard to design for embedded systems, and that some attacks are still possible.

Finally, another potential solution for intrusion detection in embedded controllers is based on power fingerprinting [35]. It consists of a physical sensor through which is possible to analyse and collect statistical data about power consumption and electromagnetic emissions, determining whether or not the process deviates from the expected operation model. Even though it is a powerful mechanism and does not interfere with critical operations, it provides limited support for forensic analysis once the attack has been detected, so this technique should be applied as a part of a security solution.

2.2.3 Control flow integrity

Control flow hijacking is one of the most used attacks to computer systems, because it usually leverages programming errors that are actually much more common than they should be. As more and more sophisticated control flow modification techniques were discovered, new defenses have been proposed, but only some of them are applicable to embedded systems. In 2012 Reeves et al. [36] proposed Autoscopy Jr., an intrusion detection system designed for embedded systems, which is focused on detecting control flow alterations instead of malicious code insertions. Its approach consists of two phases: the learning phase and the detection phase. During the learning phase it collects control flow information about the function pointers used by the kernel, building a data structure which will be used in the second phase. The data structure basically maintains, for each monitored function pointer, a list of function addresses reachable from that pointer together with other control flow information. During the detection phase it continues monitoring direct and indirect calls, and it generates an alert if an unknown function is reached from a monitored function pointer.

Habibi et al. [37] proposed a defensive technique for ARM architecture, named DisARM, effective against both code-injection and code-reuse attacks. Relying on the assumption that buffer overflow attacks lead the execution to a different return address than expected, they designed a mechanism for verifying the actual return address at runtime, thus protecting any potentially vulnerable program. First, they look for all the critical instructions contained into the program, defined as the ones that take input from the stack and affect the program counter directly or indirectly (e.g. through the link register). Second, they modify the binary code by putting a verification block before each critical instruction, so that the execution is stopped whenever a control flow manipulation is attempted through the stack.

Many other control flow integrity (CFI) solutions for embedded systems rely on hardware modifications in order to require smaller overheads. Francillon et al. [38] presented a technique to protect low-cost embedded systems against malicious manipulation of their control flow by using a simple hardware modification to divide the stack in a data and a control flow stack (or return stack). The access to the control flow stack is restricted only to return and call instructions, thus implementing an Instruction Based Memory Access Control (IBMAC) in hardware. Abad et al. [39] proposed a hardware-based security approach with predictable overhead for embedded real-time systems. They perform CFI checks on a real-time task by adding an On-chip Control Flow Monitoring Module (OCFMM) to the processor core with its own isolated memory unit. OCFMM monitors the run-time control flow and compares it to the stored Control Flow Graph determined in advance. Davi et al. [40] designed novel security hardware mechanisms to enable fine-grained CFI checks, based on three security policies. First, each function call enforce the processor to switch to a new state in which the only accepted instruction is a CFI instruction, thus restricting function calls to only target valid function entry points. Second, return instructions can only target a valid return of a function whose CFI instruction is active. The CFI instructions are identified by labels, managed through a CFI Label State Table embedded in the program data memory. Third, behavioural heuristics are used to cover typical patterns of ROP attacks. Afterwards, they provided an implementation for Intel Siskiyou Peak and SPARC, named HAFIX [41], demonstrating its security and efficiency in code-reuse protection while incurring only 2% performance overhead. Another hardware-based approach has been presented by Das et al. [42]: a fine-grained CFI at a basic block level, named basic block CFI (BB-CFI). A basic block is defined as a sequence of instructions, having a single entry and a single exit point. The policies used by BB-CFI are defined as follows: first, each function call can only target the first basic block of the function; second, each return can only target the basic block following the function call; third, indirect jumps must target a starting address of a basic block. Their approach is divided into two steps: 1) offline profiling of the program to collect control flow information data, and 2) runtime control flow checking. The control flow checker has been implemented on FPGA as a proof-of-concept, showing $< 1\%$ performance overhead, a small dynamic power consumption and a very small area footprint.

Finally, the attestation mechanism could be used to address control flow integrity as well. Abera et al. [43] presented the design and the implementation of Control-FLow ATtestation (C-FLAT), based on ARM TrustZone hardware security extensions. In their model, a verifier wants to attest runtime control flows of an application module on a remote embedded system, the prover. First, the verifier has to generate the Control Flow Graph from a clean binary of the application, storing the measurements of all the possible control-flow paths. Then, it sends a challenge to the prover, containing the name of the application and a nonce. The prover executes the application, computes a digital signature over the challenge and the executed path, and sends it back to the verifier for validation.

Chapter 3

Pin Control Attack

Before describing Pin Control Attack, a deeper analysis of the architecture of the target system is needed. Note that, although this paper is focused on PLCs, the architecture described in the next section is still valid for almost any embedded system. After having a proper knowledge of the underlying architecture, we can go deep into the description of the attack. We consider the idea behind the attack, showing how it is able to evade the currently available detection mechanisms. Next, we go further extending the applicability to a real Programmable Logic Controller, describing its architecture in more detail. Based on our architecture analysis and our tests, we can finally demonstrate that the attack is actually viable on real PLCs, even on a higher abstraction level.

3.1 Embedded architecture

As briefly reported into Chapter 1, PLCs use I/O interfaces to communicate with sensors and actuators, and in general with any external device. Digging into their architecture, we know that PLCs are usually based on a so called System on Chip (SoC). A SoC is basically an integrated circuit made of a microprocessor, a memory block and a set of peripheral controllers all enclosed together in the same chip substrate. Thus, the SoC technology provides fully capable computers having both very small size and low power-consumption. A SoC usually comes with a set of connections, also known as *pins*, usually soldered to a printed circuit board to facilitate interconnection with external modules. Many types of pins may be included in a SoC, having different purposes (power, clock, I/O, etc.). An example of such a system is the Raspberry Pi board shown in Fig. 3.1, based on a Broadcom System on Chip. Actually almost all of the embedded systems use a SoC with

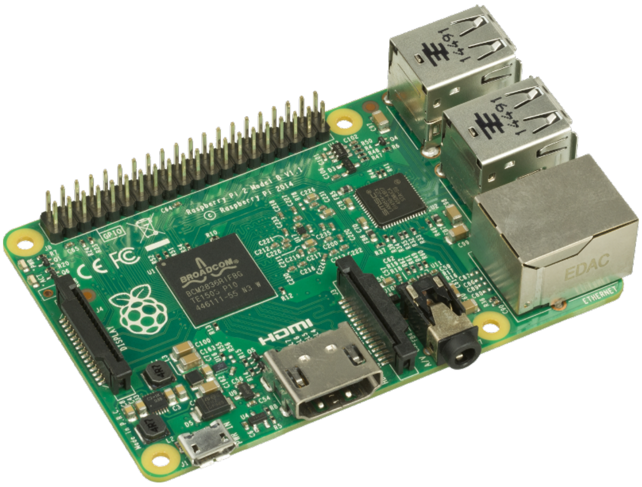


Figure 3.1: Raspberry Pi [1] with Broadcom System on Chip

similar boards, each one with its own size and configuration.

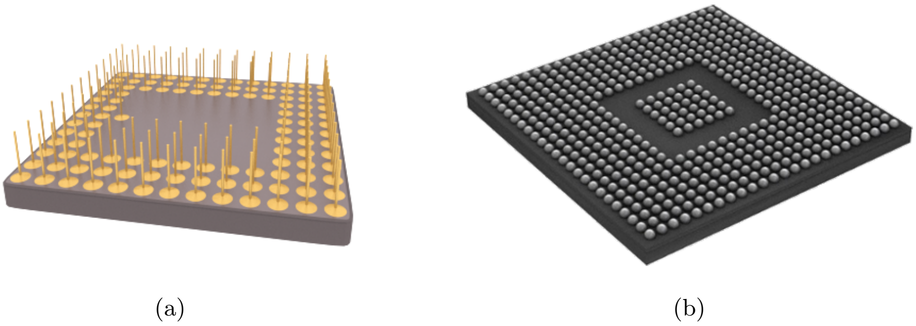
In order to accommodate different potential implementations, each pin (or group of pins) of a SoC may have multiple configuration and operating modes, depending on the board they are attached to. The configuration of the pins is managed by the pin controller, a particular subsystem of any SoC. Through the pin controller, the system can configure the operating mode of the pins, such as their input or output mode. The features of a pin controller can be grouped into two main categories:

- pin configuration: allows the system to change some electrical properties of the pins, such as direction, event detect, interrupt, etc.;
- pin multiplexing: each pin of the SoC may have many usage usages, also known as *alternate functions*, depending on what is needed by the external board. The pin multiplexing feature enables the system to specify which type of function is needed on each pin.

As the I/O attack is a very low-level attack, it is necessary to dig further into the electrical world to know how these I/O interfaces work.

3.1.1 SoC pins

I/O interfaces of a System on Chip provide a connection between internal modules and external electronic devices. As shown in Fig. 3.2, they are physically visible from the outside of the chip package, usually in the form of pins (a) or soldering balls (b).



(a)

(b)

Figure 3.2: I/O connections packaged as (a) Pin Grid Array and (b) Ball Grid Array

Internally they are connected to the silicon die through bonding wires, and are managed by a specific I/O circuit which may vary according to the specific chip. Although there are many different implementations, almost all of the available SoCs have I/O ports with very similar functionalities. For our purpose, we can describe them in an implementation-independent manner by using simplified generic schematics.

Pin configuration

The schematic depicted in Fig. 3.3 helps us to discuss about the first set of features: pin configuration. The operation mode described in this section is also known as General Purpose I/O (GPIO).

Apart from the protection diodes that serve as shield against input currents lower than V_{SS} or higher than V_{DD}, the circuit is divided into two different parts: one for output and one for input.

- **Output driver**: The output module is basically a buffer controlled by an output enable signal. This signal controls the direction of the pin (input or output). If it is high, then the pin is in output mode and the value coming from a write operation goes through the buffer to the actual pin. If it is low, the pin is in input mode and the write signal is blocked, so it is not possible to change the external value of the pin from inside anymore.

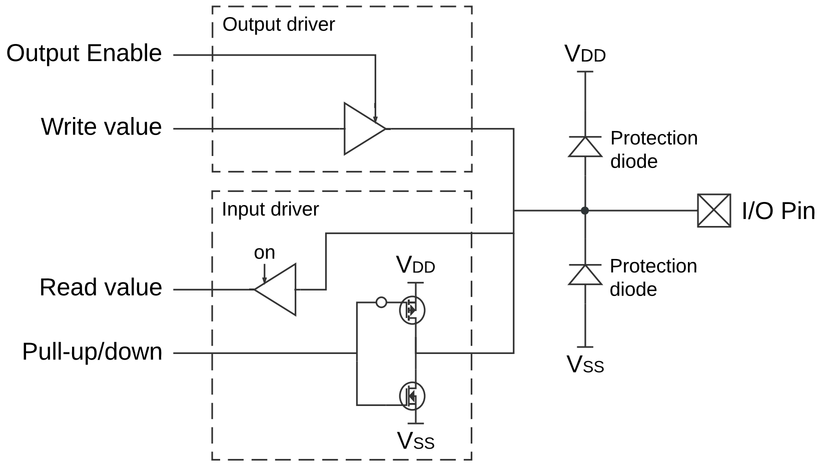


Figure 3.3: General Purpose I/O pin configuration circuit

- **Input driver**: The input driver has a similar buffer to read the value, usually having hysteresis capability which is useful for filtering unstable external values. Since the read buffer is always active, the input value is always available, even if the pin is currently working in output mode. The reason for this is merely physical: even if the external value was blocked by the buffer, one would always get a value by reading the input signal, because a value is nothing but an interpretation of the voltage level on a wire. When the pin is set as input, the pull-up/pull-down network enables the user to have a "default" value on the pin, namely a defined state maintained while the pin is not actively driven from outside. This feature is useful to avoid so called "floating" inputs.

For Pin Control Attack, what is more interesting about the circuit in Fig. 3.3 are the following two properties:

- there is no checking about the input state, so it is possible to perform a read even when the pin is in output mode;
- vice versa, it is not possible to write to a pin which has been configured as output.

Furthermore, it is also possible to drive the pull-up/down network, factually disturbing the real value of the I/O pin in an unpredictable way. In this case the effects strongly depends on the actual implementation of the printed circuit board as well as on the external components connected to the I/O pin.

Pin multiplexing

Inside the chip an I/O pin may be connected to more than one device, which can be selected depending on the application, that is the way of soldering or wiring the package into an electronic board. This SoC feature is known as pin multiplexing (also ball multiplexing, pad multiplexing, alternate functions). Even though pin multiplexing is designed for hardware configuration, in almost all modern chips it is possible to change the function at runtime.

Fig. 3.4 shows a possible hardware implementation of pin multiplexing. The I/O pin in the figure is connected to two different peripherals inside the chip, namely A and B, and it is also accessible through basic GPIO as described in Section 3.1.1 above. The access to the GPIO output driver is regulated by two in cascade multiplexers for each signal of the module. If the peripheral A is enabled, then the signals driven by A go through the multiplexers and can drive the actual value of the pin, while GPIO and peripheral B output signals are blocked. Instead, if

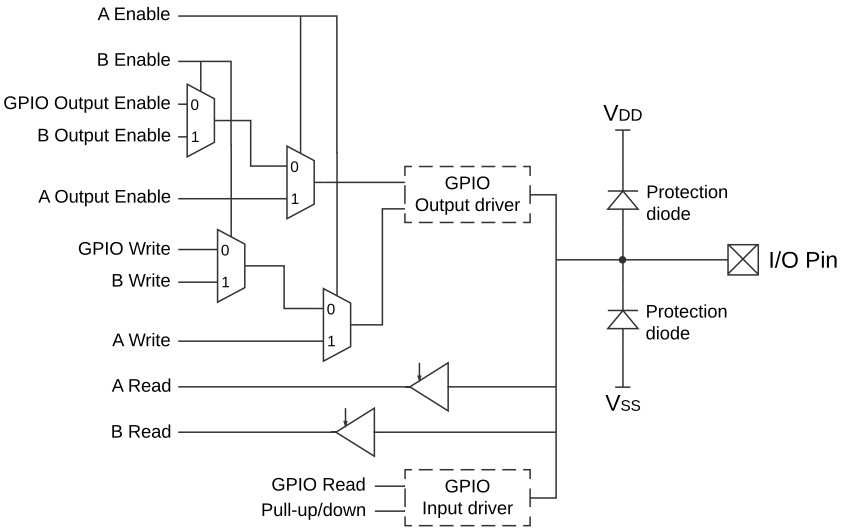


Figure 3.4: Generic I/O pin multiplexing circuit

only peripheral B is enabled then only its signals are able to reach the I/O pin. Note that in this last case the peripheral A should not be enabled, because A multiplexers have precedence against B ones. Thus, the cascading of multiplexers actually implements a priority mechanism between peripherals A and B. If neither A nor B are enabled, then no alternate function is active for the I/O pin and it could be driven by GPIO signals. Each peripheral may also have its own dedicated input line, to get values from the I/O port in the same way as GPIO does.

For our purpose, at least two interesting properties can be inferred from pin multiplexing schematic of Fig. 3.4:

- it is possible to block output signals from peripherals by simply changing the multiplexing configuration;
- the GPIO value can be read at any time, independently from the current multiplexing state of the output.

3.2 Pin Control Attack analysis

Due to our previous discussion, we can argue that Pin Control Attack is actually applicable on any System on Chip having the architecture described in Section 3.1. However, PLCs represent a particularly interesting target among all the embedded systems based on SoC. PLCs leverage the interfaces of a SoC to interact with sensors and actuators, having direct effects on our physical world. Therefore, if the I/O of a PLC is compromised, this constitutes a much more significant security and safety risk with respect to other non-critical embedded system.

As described in [2], Pin Control Attack is designed to evade off-the-shelf host-based detecion systems. Furthermore, differently from other attacks, its target is neither the firmware nor the logic of the PLC. The idea of Pin Control Attack is that the attacker can operate at the lowest level possible, targeting the interaction between the firmware and the PLC I/O, as shown in Fig. 3.5. For this reason it is also known as I/O attack. Despite its crucial function in embedded systems, the I/O hardware architecture and the I/O drivers are basically designed without any concept of security, assuming that I/O is reliable. Unfortunately, given the properties inferred in Section 3.1 from generic hardware architecture, this is often not the case.

In the following section we focus on the design of I/O attack. First, we summarise the host-based detection mechanisms taken into account during the attack design and the techniques used

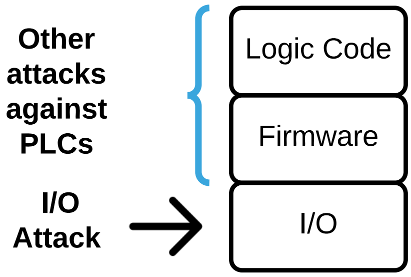


Figure 3.5: Pin Control Attack target, from [2]

to evade them. Second, we analyse the attack itself in more detail. After the design part, we describe different implementations of the attack coming from our experiments, showing how the attack works in practice. The goal of this chapter is to help us having a better understanding of the attack, and to provide a detailed threat model on which our defense design will be based on.

3.2.1 Design

Based on what is available in literature, Pin Control Attack has been designed to be different from previous attack techniques and to circumvent the off-the-shelf host-based detection systems. The authors of I/O attack have identified two main defensive mechanisms whose properties makes them more easily applicable to PLCs. We briefly analyse these defenses and show how they can be circumvented by Pin Control Attack, and then we describe the Pin Control Attack technique. Our purpose here is to have basic definitions of the concepts used by the attack, see [2] for a more detailed description.

Defense analysis

TODO Defense analysis

Attack technique

TODO Attack technique

3.2.2 Implementation

TODO Implementation.

Chapter 4

Pin Control Defense

TODO Pin Control Defense design and implementation.

Chapter 5

Experimental Results

TODO Experimental Results.

Chapter 6

Conclusions

TODO Conclusions.

Bibliography

[1] "Raspberry Pi", https://www.raspberrypi.org/.

[2] A.Abbasi, M.Hashemi, E.Zambon, S.Etalle, "Stealth Low-Level Manipulation of Programmable Logic Controllers I/O By Pin Control Exploitation", TODO Complete citation CRITIS Conference 2016.

[3] "Pin Control Subsystem", https://www.kernel.org/doc/Documentation/pinctrl.txt.

[4] N.Falliere, L.O Murchu, E.Chien, "W32.Stuxnet Dossier", Version 1.4, Febr. 2011. Online: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[5] R.Grandgenett, W.Mahoney, R.Gandhi, "Authentication Bypass and Remote Escalated I/O Command Attacks", CISR '15: Proceedings of the 10th Annual Cyber and Information Security Research Conference, Oak Ridge, Tennesse (USA), April 7-9, 2015, DOI 10.1145/2746266.2746268.

[6] D.Papp, Z.Ma, L.Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy" Privacy, Security and Trust (PST) 13th Annual Conference, Izmir (Turkey), July 21-23, 2015 pp. 145-152, DOI 10.1109/PST.2015.7232966.

[7] Z.Basnight, J.Butts, J.Lopez Jr., T.Dube, "Firmware modification attacks on programmable logic controllers", International Journal of Critical Infrastructure Protection, Volume 6, Issue 2, June 2013, pp. 76-84, DOI 10.1016/j.ijcip.2013.04.004.

[8] D.Peck, D.Peterson, "Leveraging ethernet card vulnerabilities in field devices", Proceedings of the SCADA Security Scientific Symposium, Miami Beach, Florida (USA), Jan. 18-19, 2009, pp. 1-19.

[9] A.Cui, M.Costello, S.J.Stolfo, "When Firmware Modifications Attack: A Case Study of Embedded Exploitation", 20th Annual Network & Distributed System Security Symposium, San Diego, California (USA), Febr. 24-27, 2013, DOI 10.7916/D8P55NKB.

[10] S.McLaughlin, "On Dynamic Malware Payloads Aimed at Programmable Logic Controllers", In 6th USENIX Workshop on Hot Topics in Security, 2011.

[11] S.McLaughlin, P.McDaniel, "SABOT: Specification-based Payload Generation for Programmable Logic Controllers", CCS '12: Proceedings of the 2012 ACM conference on Computer and Communications Security, New York (USA), Oct. 16-18, 2012, pp. 439-449, DOI 10.1145/2382196.2382244.

[12] D.Beresford, "Exploiting Siemens Simatic S7 PLCs", Black Hat USA 2011, Las Vegas, Nevada (USA), Aug. 3-4, 2011.

[13] J.Klick, S.Lau, D.Marzin, J.Malchow, V.Roth, "Internet-facing PLCs as a Network Backdoor", Proceedings of IEEE Conference on Communications and Network Security (CNS), Florence (Italy), Sept. 28-30, 2015, pp. 524-532, DOI 10.1109/CNS.2015.7346865.

[14] ICS-CERT, "Schneider Electric Modicon M340 Buffer Overflow Vulnerability", Dec. 17, 2015. Online: https://ics-cert.us-cert.gov/advisories/ICSA-15-351-01.

[15] ICS-CERT, "Rockwell Automation Micrologix 1100 and 1400 PLC Systems Vulnerabilities (Update A)", Oct. 27, 2015. Online: https://ics-cert.us-cert.gov/advisories/ICSA-15-300-03A.

[16] ICS-CERT, "Rockwell Automation MicroLogix 1100 PLC Overflow Vulnerability", Jan. 26, 2016. Online: https://ics-cert.us-cert.gov/advisories/ICSA-16-026-02.

[17] ICS-CERT, "Eaton ELCSoft Programming Software Memory Vulnerabilities", June 30, 2016. Online: https://ics-cert.us-cert.gov/advisories/ICSA-16-182-01.

[18] P.Chen, X.Xing, B.Mao, L.Xie, “Return-oriented rootkit without returns (on the x86)” ICICS 2010: 12th International Conference on Information and Communications Security, Barcelona (Spain), Dec. 15-17, 2010, pp. 340-354, DOI 10.1007/978-3-642-17650-0_24.

[19] S.Checkoway, L.Davi, A.Dmitrienko, A.Sadeghi, H.Shacham, M.Winandy, “Return-Oriented Programming without Returns”, CCS ’10: Proceedings of the 17th ACM conference on Computer and Communications Security, Chicago, Illinois (USA), Oct. 4-8, 2010, DOI 10.1145/1866307.1866370.

[20] R.E.Johnson, “Survey of SCADA Security Challenges and Potential Attack Vectors”, ICITST 2010: International Conference for Internet Technology and Secured Transactions, London (UK), Nov. 8-11, 2010, pp. 80-85.

[21] B.Miller, D.Rowe, “A survey SCADA of and critical infrastructure incidents”, RIIT ’12: Proceedings of the 1st Annual conference on Research In Information Technology, Calgary, Alberta (Canada), Oct. 10-13, 2012, pp. 51-56, DOI 10.1145/2380790.2380805.

[22] G.P.H.Sandaruwan, P.S.Ranaweera, V.A.Oleshchuk, “PLC Security and Critical Infrastructure Protection”, ICIIS 2013: IEEE 8th International Conference on Industrial and Information Systems, Peradeniya (Sri Lanka), Dec. 17-20, 2013, pp. 81-85, DOI 10.1109/ICIInfS.2013.6731959.

[23] A.Clark, Q.Zhu, R.Poovendran, T.Baar, “An Impact-Aware Defense against Stuxnet”, ACC 2013: 1st American Control Conference, Washington, DC (USA), June 17-19, 2013, pp. 4140-4147, DOI 10.1109/ACC.2013.6580475.

[24] D.Hadiosmanovi, R.Sommer, E.Zambon, P.H.Hartel, “Through the Eye of the PLC: Semantic Security Monitoring for Industrial Processes”, ACSAC ’14: 30th Annual Computer Security Applications Conference, New Orleans, LA (USA), Dec. 08-12, 2014, pp. 126-135, DOI 10.1145/2664243.2664277.

[25] ARM Limited, “ARM Security Technology - Building a Secure System using TrustZone Technology”, 2009. Online: `http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf`.

[26] P.Koeberl, S.Schulz, A.Sadeghi, V.Varadharajan, “TrustLite: A Security Architecture for Tiny Embedded Devices”, EuroSys ’14: Proceedings of the Ninth European Conference on Computer Systems, Amsterdam (Netherlands), April 13-16, 2014, DOI 10.1145/2592798.2592824.

[27] A.Fuchs, C.Krauß, J.Repp, “Advanced Remote Firmware Upgrades Using TPM 2.0”, in the book “ICT Systems Security and Privacy Protection: 31st IFIP TC 11 International Conference, SEC 2016, Ghent, Belgium, May 30 - June 1, 2016, Proceedings” edited by J.Hoepman, S.Katzenbeisser, Springer International Publishing, 2016, pp. 276-289, DOI 10.1007/978-3-319-33630-5_19.

[28] B.Lee, J.Lee, “Blockchain-based secure firmware update for embedded devices in an Internet of Things environment”, The Journal of Supercomputing, 2016, pp. 1-16, DOI 10.1007/s11227-016-1870-0.

[29] X.Wang, C.Konstantinou, M.Maniatakos, R.Karri, S.Lee, P.Robison, P.Stergiou, S.Kim, “Malicious Firmware Detection with Hardware Performance Counters”, IEEE Transactions on Multi-Scale Computing Systems, Vol. 2, Issue 3, July-Sept. 2016, pp. 160-173, DOI 10.1109/TMSCS.2016.2569467.

[30] S.Zonouz, J.Rrushi, S.McLaughlin, “Detecting Industrial Control Malware Using Automated PLC Code Analytics”, IEEE Security & Privacy, Vol. 12, Issue 6, Nov.-Dec. 2014, pp. 40-47, DOI 10.1109/MSP.2014.113.

[31] L.Garcia, S.Zonouz, D.Wei, L.P.de Aguiar, “Detecting PLC Control Corruption via On-Device Runtime Verification”, Resilience Week (RWS), Chicago, Illinois (USA), Aug. 16-18, 2016, pp. 67-72, DOI 10.1109/RWEEK.2016.7573309.

[32] A.Cui, S.J.Stolfo, “Defending Embedded Systems with Software Symbiotes”, in the book “Recent Advances in Intrusion Detection: 14th International Symposium, RAID 2011, Menlo Park, CA, USA, September 20-21, 2011. Proceedings”, edited by R.Sommer, D.Balzarotti, G.Maier, Springer Berlin Heidelberg, 2011, pp. 358-377, DOI 10.1007/978-3-642-23644-0_19.

[33] A.Seshadri, A.Perrig, L.Doom, P.Khosla, “SWATT: SoftWare-based ATTestation for embedded devices”, Proceedings of IEEE Symposium on Security and Privacy, Oakland, California (USA), May 9-12, 2004, pp. 272-282, DOI 10.1109/SECPRI.2004.1301329.

[34] C.Castelluccia, A.Francillon, D.Perito, C.Soriente, “On the difficulty of software-based attestation of embedded devices”, CCS’09: Proceedings of the 16th ACM Conference on Computer

and Communications Security, Chicago, Illinois (USA), Nov. 9-13, 2009, pp. 400-409, DOI 10.1145/1653662.1653711.

[35] C.A.Gonzalez, A.Hinton, “Detecting Malicious Software Execution in Programmable Logic Controllers Using Power Fingerprinting”, in the book “Critical Infrastructure Protection VIII: 8th IFIP WG 11.10 International Conference, ICCIP 2014, Arlington, VA, USA, March 17-19, 2014, Revised Selected Papers”, edited by J.Butts, S.Shenoi, Springer Berlin Heidelberg, 2014, pp. 15-27, DOI 10.1007/978-3-662-45355-1_2.

[36] J.Reeves, A.Ramaswamy, M.Locasto, S.Bratus, S.Smith, “Intrusion detection for resource-constrained embedded control systems in the power grid”, International Journal of Critical Infrastructure Protection, 2012, Vol. 5, No. 2, pp. 74-83, DOI 10.1016/j.ijcip.2012.02.002.

[37] J.Habibi, A.Panicker, A.Gupta, E.Bertino, “DisARM: Mitigating Buffer Overflow Attacks on Embedded Devices”, in the book “Network and System Security: 9th International Conference, NSS 2015, New York, NY, USA, November 3-5, 2015, Proceedings”, edited by M.Qiu, S.Xu, M.Yung, H.Zhang, Springer International Publishing, 2015, pp. 112-129, DOI 10.1007/978-3-319-25645-0_8.

[38] A.Francillon, D.Perito, C.Castelluccia, “Defending Embedded Systems Against Control Flow Attacks”, SecuCode ’09: Proceedings of the first ACM workshop on Secure execution of untrusted code, Chicago, Illinois (USA), Nov. 9, 2009, pp. 19-26, DOI 10.1145/1655077.1655083.

[39] F.Abad, J.Woude, Y.Lu, S.Bak, M.Caccamo, L.Sha, R.Mancuso, S.Mohan, “On-Chip Control Flow Integrity Check for Real Time Embedded Systems”, 2013 IEEE 1st International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA), Taipei, Taiwan, Aug. 19-20, 2013, pp. 26-31, DOI 10.1109/CPSNA.2013.6614242.

[40] L.Davi, P.Koeberl, A.Sadeghi, “Hardware-Assisted Fine-Grained Control-Flow Integrity: Towards Efficient Protection of Embedded Systems Against Software Exploitation”, DAC ’14: Proceedings of the 51st Annual Design Automation Conference, San Francisco, California (USA), June 1-5, 2014, pp. 133:1-133:6, DOI 10.1109/DAC.2014.6881460.

[41] L.Davi, M.Hanreich, D.Paul, A.Sadeghi, P.Koeberl, D.Sullivan, O.Arias, Y.Jin, “HAFIX: Hardware-Assisted Flow Integrity Extension”, DAC ’15: Proceedings of the 52nd Annual Design Automation Conference, San Francisco, California (USA), June 7-11, 2015, pp. 74:1-74:6, DOI 10.1145/2744769.2744847.

[42] S.Das, W.Zhang, Y.Liu, “A Fine-Grained Control Flow Integrity Approach Against Runtime Memory Attacks for Embedded Systems”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 24, No. 11, Nov. 2016, pp. 3193-3207, DOI 10.1109/TVLSI.2016.2548561.

[43] T.Abera, N.Asokan, L.Davi, J.Ekberg, T.Nyman, A.Paverd, A.Sadeghi, G.Tsudik, “C-FLAT: Control-FLow ATtestation for Embedded Systems Software”, CCS ’16: Proceedings of the 23rd ACM Conference on Computer and Communications Security, Vienna (Austria), Oct. 24-28, 2016, TODO Complete citation: conference in progress.